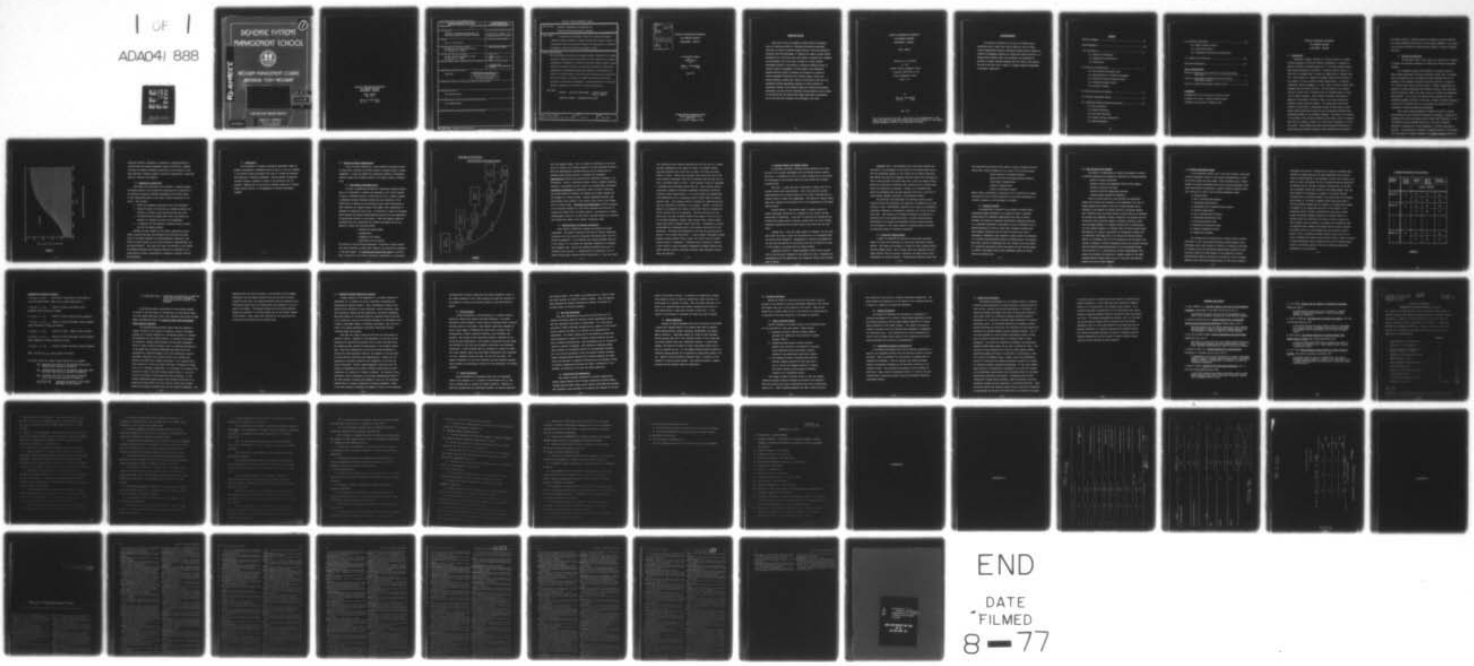
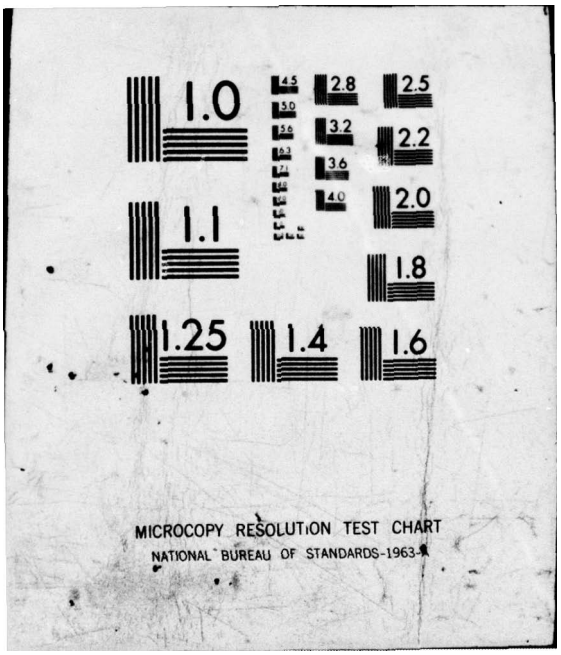
AD-A041 888 DEFENSE SYSTEMS MANAGEMENT COLL FORT BELVOIR VA F/G 9/2
TECHNICAL PERFORMANCE MEASUREMENT FOR COMPUTER SOFTWARE DEVELOP--ETC(U)
MAY 74 M A BUCCIARELLI

UNCLASSIFIED NL

1 OF 1
AD A041 888



END
DATE FILMED
8-77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DEFENSE SYSTEMS MANAGEMENT SCHOOL

PROGRAM MANAGEMENT COURSE
INDIVIDUAL STUDY PROGRAM

FORT BELVOIR, VIRGINIA 22060

BUCCIARELLI

TECHNICAL PERFORMANCE MEASUREMENT FOR COMPUTER SOFTWARE DEVELOPMENT PROGRAMS

STUDY REPORT
PMC 74-1

Marco A. Bucciarelli
Lt. Col. USAF

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* TECHNICAL PERFORMANCE MEASUREMENT FOR CONPUTER SOFTWARE DEVELOPMENT PROGRAMS	5. TYPE OF REPORT & PERIOD COVERED Study Project Report 74-1 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Marco A. Bucciarelli	8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS DEFENSE SYSTEMS MANAGEMENT COLLEGE FT. BELVOIR, VA 22010	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DEFENSE SYSTEMS MANAGEMENT COLLEGE FT. BELVOIR, VA 22060	12. REPORT DATE 74-1 13. NUMBER OF PAGES 58
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

UNLIMITED

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

SEE ATTACHED SHEET

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

SEE ATTACHED SHEET

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

DEFENSE SYSTEMS MANAGEMENT SCHOOL

STUDY TITLE: TECHNICAL PERFORMANCE MEASUREMENT FOR COMPUTER SOFTWARE DEVELOPMENT PROGRAMS

STUDY GOALS: To examine the development cycle for contractor produced computer software intended to support operational computer systems. To investigate current and proposed methods for measurement of technical performance during the software development cycle.

STUDY REPORT ABSTRACT

The development of software in computer systems acquisition is rapidly becoming a limiting factor with regard to cost, schedule and technical performance. Following an overview of relevant factors related to the software development process, the question of technical performance measurement is specifically addressed. Methods of establishing the status of software design, code and test are discussed in conjunction with the concept of utilizing an integrated software engineering approach to performance measurement. Alternate methods of monitoring software programs are presented for future consideration.

KEY WORDS:

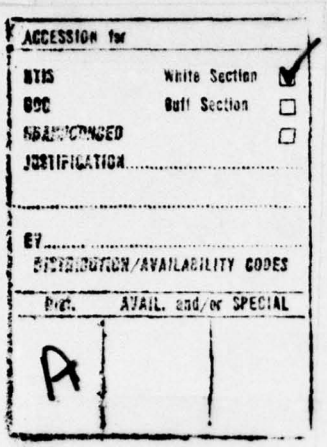
MATERIEL DESIGN AND DEVELOPMENT COMPUTER PROGRAMS
QUALITY CONTROL

COMPUTER SOFTWARE PERFORMANCE EVALUATION

NAME, RANK, SERVICE
Marco A. Bucciarelli, Lt. Col., USAF

CLASS
PMC 74-1

DATE
May 1974



TECHNICAL PERFORMANCE MEASUREMENT FOR COMPUTER SOFTWARE DEVELOPMENT PROGRAMS

An Executive Summary
of a
Study Report
by

Marco A. Bucciarelli
Lt. Co. USAF

May 1974

Defense Systems Management School
Program Management Course
Class 74-1
Fort Belvoir, Virginia 22060

EXECUTIVE SUMMARY

This paper traces the computer program software development cycle and examines methods for technical performance measurement which may be applied by System Program Offices. It has been generally recognized that the development of software for modern computer system acquisitions is a limiting factor with regard to program cost, schedule, and performance. One of the major problems in a large software development project is the ability to portray an accurate picture of software status to management. In this regard a two dimensional weighted software matrix is examined and proposed as a method to improve management visibility into software design, coding, and testing progress. Another major theme of this paper is the use of an integrated software engineering approach to focus attention on fundamental aspects of the software design for technical performance measurement. In this case the designated characteristics can be related to the qualitative and quantitative system performance requirements and can be monitored throughout the development life cycle.

TECHNICAL PERFORMANCE MEASUREMENT FOR COMPUTER SOFTWARE DEVELOPMENT PROGRAMS

STUDY REPORT

Presented to the Faculty
of the
Defense Systems Management School
in Partial Fulfillment of the
Program Management Course
Class 74-1

by

Marco A. Bucciarelli
Lt. Col. USAF

May 1974

ACKNOWLEDGEMENTS

Of particular assistance in the area of software status monitoring was Mr. Robert Kent, GS-13, USAF. Mr. Kent is Chief, Software Engineering Division, 427M System Program Office, Deputy for Command and Management Systems, Hq. Electronics System Division, L. G. Hanscom Field, Bedford, Mass. His assistance was invaluable in providing a highly informed sounding board for ideas on the subject. The use of his free time to engage in lengthy technical discussions is sincerely appreciated.

CONTENTS

List of Illustrations

Attachments

TECHNICAL PERFORMANCE MEASUREMENT FOR COMPUTER SOFTWARE DEVELOPMENT PROGRAMS

1.0 Introduction

Operational computer software is a critical portion of computer resources employed as dedicated elements, subsystems or components of systems developed or acquired under the U. S. Air Force program management concept established in AFR 800-2. Software commonly refers to a collection of programs used to extend the capabilities of computer hardware. Each program is a sequence of machine instructions necessary for the computer to solve a specific problem (application). To simplify programming, machine language is, in turn, used to extablish higher level languages such as Fortran and Cobol. As the complexity of computers increase, another form of software called an operating system or executive is usually required. Such software is designed specifically to manage all the hardware resources as well as the other applications software residing within the system. The terminology "operational" software is used herein to distinguish between military weapon system weapon system applications versus the more standardized automatic data processing programs of the business community. The latter is but sub-set of the former in the military operational environment, where the requirement exists to acquire, process, and control data, frequently on a real-time basis. These demands may require multi-processor applications to handle heavy and variable workloads, often originating from scattered

and remote locations. Examples range from command and control functions of the World Wide Military Command & Control System (WWMCCS) to the targeting and launch functions of discrete weapon systems as Trident, Minuteman, and SAM-D.

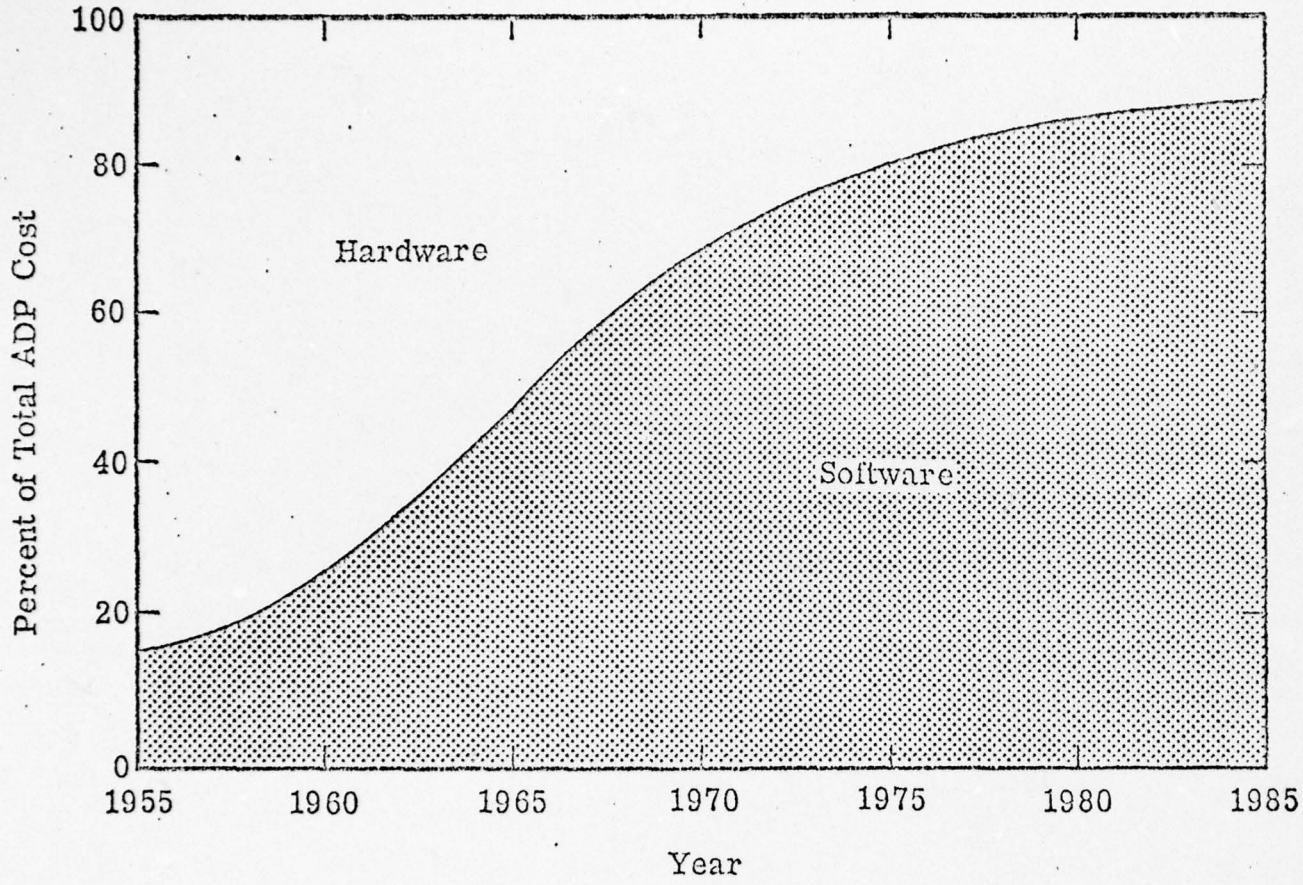
1.1 Purpose and Background

The principal theme of this paper is an examination of methods to measure the technical performance of operational software development programs.

It is generally accepted that software is a critical aspect of weapon system performance and recent disturbing trends[1] (Fig. 1) indicate that the acquisition cost of software has begun exceeding the cost of hardware in most computer installations and are projected to represent up to 80% of total costs in the next decade. Further, unexpectedly low measures of quality experienced with delivered software, have resulted in direct increases in system life-cycle-costs due to growth and maintenance difficulties[2]. For at least one large Air Force command and control acquisition program, familiar to the author, software currently represents over half of the program costs and is presently the critical factor to achievement of system performance requirements.

The overall program management of computer resources in system acquisitions has been officially recognized as presenting a unique set of problems by the Air Force Systems Control Project ACE[3] (Findings 31 & 42). The problem is being addressed directly by a new draft regulation AFR 800-14 (Attachment #1) titled "Management of Computer Resources in Systems". The regulation is intended to establish policy for the acquisition and support of computer equipment and computer programs employed as

FIGURE 1



Hardware/Software Cost Trends

dedicated elements, subsystems or components of systems developed or acquired under the program management concept in AFR 800-2. Computer resources in systems as defined by AFR 800-14 is the totality of computer equipment, computer programs, associated documentation, contractual services, personnel and supplies.

1.2 Assumptions & Limitations

This paper will focus only on the software or computer program development portion of the acquisition process and will concentrate on the problem of measuring technical performance after contract award. In thus limiting the scope of this paper, certain assumptions are implicit. They include:

1. System and subsystem requirements have been established;
2. Trade-off studies for selecting and optimizing hardware and software to satisfy a given application have been made;
3. Cost-effectiveness analyses which obtain the desired performance at a minimum cost have been accomplished;
4. A contractor has been selected to design, develop, produce, and test the system software.

Software has been singled out for further examination because typical systems which have been developed by the Services have been found to be largely composed of non-interchangeable components, hand crafted at great expense, and are often unreliable, unmaintainable, and and unintelligible[4]. This paper will limit its attention to the problem of software development and production which can be considered to include analysis, design, implementation, integration, testing, delivery and maintenance.

1.3 Organization

In the analysis of technical performance measurement (TPM) for software developments, consideration will be given to the major segments of the software system development life cycle to include the mechanics and psychology of computer programming. Current methods of measuring technical progress, including a survey of the literature, will be assessed. Emphasis will be placed upon potential application by System Program Offices (SPO's) in the management of software development programs.

2.0 Relevant Software Considerations

One of the major problems in a large software development project is being able to portray an accurate picture of software status to upper management. To gain an insight into prospective methods of determining software status, the following aspects of the subject require consideration.

2.1 The Software Development Cycle

In order to appreciate the nature of operational software development, it is necessary to consider the components of the overall process by which a developer/contractor produces the end item. Figure 2 depicts a generalized process including verification and validation (V & V). Verification is defined as any technique by which coded instructions or model equations are compared against some higher level requirements to determine if objectives will be met. By validation is meant any technique which includes the overall system/subsystem execution and the application of input/output analysis to the results. After the computer system requirements have been established the software development can be considered to follow the following phases[4]:

- Analysis and system design
- Implementation
- Integration and testing
- Installation and maintenance

The handbood of Data Processing Management [5] describes a similar system life cycle consisting of eight major segments, but essentially conforming to the above phases. The analysis and system design phase is initiated with a definition of the overall functional requirements to be satisfied

Validation and Verification

and The Software Development Process

FIGURE 2

with the computer system. This is followed by definition of the interaction of hardware and software required to provide necessary functions. With this understanding, software functions may be defined and the software system design approach is completed and documented. Implementation involves the activities most generally considered to bo programming. Design of individual modules, coding, language processing (assembly & compilation and unit testing are implementation activities). Integration and Testing are deceptively simple and easy to describe: the various modules of software and hardware are brought together and and debugged as a working system. Typical experience indicates that approximately fifty percent of the software costs occur in the integration and testing phase. Installation and Maintenance involves packaging of the resultant software in a form which minimuzes the difficulty of loading the software into the target computer. Demonstration of system acceptability, follow-up on latent errors, and user support are considerations during this phase of the software development cycle.

2.2 Characteristics of Computer Programming

There exists a significant variance in productivity between programmers. One study to determine efficiency of various programming techniques could only show that programmer performance might vary 26:1 between programmers[6]. It is estimated that current software costs are in the range of 15- 30 dollars per executable instruction of debugged and documented code. Typical net software production rates are 2- 5 instructions per man-hour. A program which is considered of low to medium size may have as many as 22,000 instructions. It has been found

that programming costs increase exponentially with the size of a program and when programming is done under the duress of extremely limited processor resources such as CPU time or memory, the costs can easily double or triple. Coding alone represents less than 25 per cent of the total software development costs and the learning curve has been seen to have a pronounced effect on costs. A wide variety of factors[7] contribute to programs being structured the way they are. These include: machine limitations; language limitations; programmer limitations; historical accidents; and/or specifications. As a result nearly all programs have some imperfections and to complicate matters further there are usually multiple users. Most programs should be designed for adaptability since they will inevitably be modified to some degree. However, a program designed to be adaptable will likely be inefficient while an extremely efficient program will not be adaptable. The term efficiency does not stand alone, but must be further defined with respect to one or more of the following measurers: compiler time; computer time; compiler speed; people time; and/or excessive paging (page size). Programming is usually accomplished by a programming team as the primary working unit in most situations. The nature and composition of the team are particularly essential to its relative performance. A large software development project generally brings together several teams, the control of whom is particularly crucial to management. Programmers may be required to perform a wide variety of tasks including: documentation; program library work; system testing; progress reporting; standards work; diagnostic programming; and education.

2.3 Software Testing and Quality Control

As previously indicated, software testing represents over fifty per cent of all program development costs and thereby warrents special attention. An organized test plan is a vital prerequisite to the successful development of any software system and must include the following considerations[4]:

Unit Test -- A unit test plan is developed to assure that all individual modules meet specifications. Such a test plan must contain specifications for the routines which are to be used to exercise the individual module to insure its completeness. The ability to clearly define these test routines is an excellent test of the completeness of the system specifications.

Nucleus Testing -- The minimum number of modules required to achieve meaningful interaction in a subsystem of the overall software system must be determined. A plan must be developed for integrating and testing this minimum number of modules, probably with the other software components simulated by the testing routines used during the unit test phase.

Staging Plan -- Once the nucleus system is debugged, the test plan must detail the order in which the new modules are to be interfaced with the proven system components. Requirements for testing the subsystem as it grows with the addition of new modules must also be detailed in terms of test stimuli to be applied and the desired results.

Acceptance Criteria -- System acceptance criteria must be specified as the last major component of the system test plan. Techniques for demonstrating that the system meets the acceptance criteria must be specified in detail.

Debugging Tools -- The debugging tools used should maximise the visibility to the programmer of the actions of the program under test so that the unexpected response to test stimuli may be readily identified. They should also allow the programmer absolute control over directing the flow of control within the system under test so that the suspect control paths may be exercised to determine their validity and all control paths within the system may be exercised. Further, it is essential that during the debugging phase the same response occurs from a given set of input stimuli, leading, therefore, to reproducible results.

The definition and establishment of universal quality control standards during the development and production of software has not been realized to date. Nevertheless, inspection can be made for readability, modularity, adherence to documentation standards, and the use of structured code. Code reading is an important technique for detection of low-level errors not necessarily subject to formal testing scrutiny. Unfortunately, the typical Government/SPO monitoring practice exerts little influence on the quality aspects of software during the design and production phases of software development[2].

2.4 Analytical Computer Models

A degree of research[8] has been done in developing analytical models to project the performance of large-scale time-shared computer systems. The models are stochastic in nature and their analysis usually involves queuing theory. Most of these models are concerned with subsystem behavior such as processor scheduling, secondary memory, multiprogramming, and program behavior. Although many analytical models have

been developed and analyzed, there exists a general consensus that most models have a serious weakness in one or more of the following areas:

System behavior vs subsystem behavior;

Paging activities and multiprogramming behavior;

Space-domain considerations;

Overhead considerations;

Choice of performance measure.

Models remain, nonetheless, an important tool for prediction of system performance but have limited application to the actual measurement of technical programs in the development of software.

2.5 Synthetic Programs

A straightforward approach[9] to the problem of complex multiprogramming systems evaluation is to design and model a synthetic program which will represent substantially some level of system activity. the ability to reproduce realistically computing situations in a controlled experimental environment, can be an invaluable tool in system evaluation. As with any other model, synthetic programs have their inherent limitations not the least of which is the cost of development. Nonetheless, they represent a useful method for evaluation of a software system/subsystem during the testing phase of the development cycle. Synthetic programming does not, however, hold much promise for performance measurement during the earlier and more crucial phases of software development during which fundamental design and coding efforts are taking place.

3.0. Data Collection and Analysis

The subject of programming and software development as related to systems acquisition was examined by employing the following methods:

Review of current literature;

Personal contact with responsible service staff members;

Software contractor briefings;

Discussion with SPO engineering personnel;

Personal system acquisition experience.

After an initial review of the literature and preliminary contact with Service staff agencies in the Washington, D.C. area, it became apparent that little in the way of formal guidance was in existance on the subject of software technical performance measurement. Therefore, data from these sources is used primarily to establish the setting and background. Relevant information was obtained from two contractor sources presently engaged in software activities. These were The Hughes Company, in a special briefing held at Headquarters, Air Force Systems Command on 7 February 1974; and Science Applications Incorporated, in a special presentation on independent software test and evaluation technology, held at Hq. Electronic Systems Division (AFSC), on 15 December 1973. The most significant data relating to the problem of managing a software development program was received directly from the Software Engineering Division of the 427M System Program Office located at L. G. Hanscom Field, Bedford, Massachusetts. This program is concerned with acquiring a computer system for the NORAD Cheyenne Mountain Complex which is part of the World Wide Military Command and Control System (WWMCCS).

4.0. Software Development Status

As was mentioned earlier, one of the major problems in any large software development project is the accurate representation of software status during each major phase of the development cycle. Current methods are derived principally from earlier program management experience with hardware acquisitions. A typical software milestone report will contain all or some of the following sequential events:

1. Contract Award
2. Part I Specifications Complete
3. Preliminary Design Review
4. Preliminary Part II Specifications Complete
5. Critical Design Review
6. Part II Specifications Complete
7. Preliminary Qualification Test
8. Formal Qualification Test
9. Functional Configuration Audit
10. Physical Configuration Audit
11. Segment Test Complete

The problem with the exclusive use of these hardware oriented milestones, is that they presume relatively distinct phases in the development cycle. Basic design for hardware is essentially completed in conjunction with the Preliminary Design Review, however for software basic design can extend through to the completion of PartII Specifications. Hardware fabrication or production activity normally commences after the Critical Design Review. In the case of software

development the parallel to fabrication is coding and checkout which overlaps the design phase and can begin immediately after contract award and extend through to the beginning of formal test activity. As a result of these differences, the use of existing milestone definitions and methods does not portray an accurate representation of the software development status with relation to the actual design, code, and test activity of the program. An approach to achieving improved visibility into the software status at any given point in the development process, is to utilize a weighted matrix approach by which a discrete software element such as a module or an applications program is weighted in two dimensions. The first is with respect to the individual module's relative contribution to the entire software system. The second dimension considers the software module as a discrete entity and establishes the degree of effort required for module design, code and checkout, and formal test. Essentially then the weighted matrix assigns relationships between software elements and between tasks to be accomplished for each particular element. Figure 3 shows a generalized weighted software matrix which can be used to determine the status of each software module in terms of percent of module development effort completed. By use of the system scaling factors, the relative portion of the total software system effort completed can be established.

A Generalized Weighted Software Matrix

SOFTWARE ELEMENT	SYSTEM SCALING FACTORS	PERCENT DESIGN	PERCENT CODE & CHECKOUT	PERCENT VALIDATION
		PERCENT COMPLETED		
Module #1	a_1	X_1	Y_1	Z_1
		A_1	B_1	C_1
Module #2	a_2	X_2	Y_2	Z_2
		A_2	B_2	C_2
. . . . .	. . . . .	. . . . .	. . . . .	. . . . .
Module #N	a_n	X_n	Y_n	Z_n
		A_n	B_n	C_n

FIGURE 3

EXPLANATION OF SYMBOLS IN FIGURE 3

1. $a_1, a_2, \ldots\ldots, a_n$: The relative contribution of each module to the total system effort (sum of all a_n equals 100 percent).

2. $X_1, X_2, \ldots\ldots, X_n$: Percent of module development effort estimated and/or allocated to design.

3. $A_1, A_2, \ldots\ldots, A_n$: Percent of module design actually completed.

4. $Y_1, Y_2, \ldots\ldots, Y_n$: Percent of module development effort estimated and/or allocated to coding and checkout.

5. $B_1, B_2, \ldots\ldots, B_n$: Percent of module actually coded & checked.

6. $Z_1, Z_2, \ldots\ldots, Z_n$: Percent of module development effort estimated and/or allocated to formal validation and test.

7. $C_1, C_2, \ldots\ldots, C_n$: Percent of module validation actually complete.

NOTE: The sum of X_n, Y_n, and Z_n equals 100 percent

To further clarify the concept consider Module #1 as an example:

A_1X_1 represents that portion of the required design effort which has been completed for Module #1

B_1Y_1 represents that portion of the required coding and checkout effort which has been completed for Module #1

C_1Z_1 represents that portion of the requred validation effort which has been completed for Module #1

$A_1X_1 + B_1Y_1 + C_1Z_1$ represents the percent of total module development effort completed

$a_1(A_1X_1 + B_1Y_1 + C_1Z_1)$ represents the contribution of Module #1 to the total software development effort which has been actually been completed

By performing similar calculations for each software element or module we have the percent of contribution for each element toward the total system development effort. The arithmetic sum of each of these module contributions yields the <u>percent of total system development effort actually completed</u>.

The proposed weighted software matrix does not represent a complete solution to the problem of managing a software development program, but is merely suggested as a structured approach to increase SPO visibility into the software status . It posesses the advantage of arriving at a single figure-of-merit for the entire development effort in terms of percent complete. There are unlimited variations to to the method such as further subdividing the design effort into requirements & algorithim design, or into data structures & control structures. Similarly, the validation effort can be further subdivided into module test & module integration. It is apparent that a clear definition of terms and proper allocation of weights and scaling factors is an essential prerequisite. The allocation can be made by the contractor as part of his work package planning effort, or can be allocated by the Program Office based upon prior experience. If the software development program employs work breakdown structures in accordance with MIL-STD-881, then the contractor should have internal module work packages containing both cost and schedule information. The

weighted matrix can then be tailored to the contractor's work package description, and the weights extracted from the man hours of effort planned for each task. The concept described above was attempted by the 427M System Program Office in February 1974, and attachment 2 is an example of the result. The summary findings were that the NCS Software Segment was estimated to be 51.8% complete and the SCC Software Segment 30.9% complete. These results were confirmed by the contractors and correlated with existing cost and schedule data.

5.0 Integrated Software Engineering Approach

Another approach to the measurement of a software contractor's performance is to consider the overall technology of designing and developing the computer software. Basic considerations common to any system include definition of data structures, control structures, real-time processing, program and data organization, and memory management. When these aspects of the system design are baselined either by the SPO or the contractor, they can be monitored throughout the development cycle. To yield a meaningful measure of technical performance, they can be related to the system quantitative performance requirements such as processing speed and accuracy.

Both data and control structures deal with the micro-aspect of computer software. Questions of the organization of the data and the techniques for achieving the required flow of control are dealt with. The remaining items consider the macro-aspect of the software environment. Real-time considerations involve the problems encountered in relating the basic sequential nature of the computer to the real world in which multiple activities occur simultaneously. Program and data organization techniques provide for the integration of the data and control structures. Finally, memory management topics deal with the problem of implementing this overall software system within the real constraints of a given set of computer resources. The analysis of data structures, control structures, and real-time considerations relate to the basic problem of enabling the computer to carry out its functions. Considerations of program organization and memory management relate to the basic problem of enabling the computer to carry out its functions.

Considerations of program organization and memory management relate to the overall question of how to best organize and apply the resources of the computer to execute the problem's solution in the most effective manner.

5.1 Data Structures

A vital prerequisite to the development of a computer software system is a careful and complete data-base definition. The system data-base is that collection of data items and structures which constitute the data portion of the system. Most computer application programs involve a large number of different data types such as integer, real or floating-point, logical, character variables, state variables, and pointers. The concept of data structures provides a framework for relating the data requirements to the problem of organizing the data within the processor so that they may be updated, referenced, and manipulated efficiently. A data structure then is a grouping of a number of simple data items together, which have some logical significance when considered as a set. Examples include the array, the list and the queue. The logical organization of such data structures and their physical organization in memory are particularly important for the achievement of efficient programs.

5.2 Control Structures

Control structures are techniques which alter the sequential nature of the computer, i.e., execution of instructions one at a time from an ordered list, to achieve the desired operation. Examples include the unconditional and conditional transfer, the typical iteration,

and decision tables. The concept of the subroutine is a control structure which provides the basis for modular software. When the required data structures and control structures are defined, the design of a software module is virtually complete.

5.3 Real Time Processing

Real time considerations may involve the efficient use of a computer with memory speed and execution cycles significantly faster than the peripherals with which it communicates. Or it may involve situations in which individual programs are not capable of fully utilizing the computer and there is no conflict in the requirement for peripherals. Yet another instance may be the situation in which different programs are required for different system functions and each must be executed simultaneously. Such requirements lead to concepts of tasking or parallel processing. This requires an operating system which provides for the general function of selecting which programs are to be run and allocating to them the required computer resources. Typical operating system functions will include scheduling and supervising the execution of programs, allocating main and secondary storage as required, supervising interrupts and events, coordination of the programs, and handling of the input and output operations.

5.4 Program and Data Organization

This subject concerns techniques for internal communication between program modules such as through permanently allocated memory, through blank and named common, and/or through files. When many programs must reference a data structure it is usually left resident in the main

memory at some fixed location. An extension of communication through fixed memory is by use of named or unnamed data blocks allocated in a fixed manner by a compiler or loader. When the systen data base is too large to be contained within main memory, components are stored on a bulk memory device in named data structures called files. A file system is then provided to reference and obtain this data on demand.

5.5 Memory Management

Concepts of memory management deal with the problem of allocating a shared main computer memory to the program which must be executed. Memory sharing methods include overlaying, partitioning, and dynamic memory allocation. Overlaying schemes are simple but may compromise system response times due to the repetitive nature of the overlay function within modules which have constituent overlays. In partitioned systems, the memory is divided into fixed length areas within which a group of programs may run. For the dynamic memory-allocation system, an executive program searches for an application program which wishes to be run. Memory is then allocated at execution time, if required, at the expense of a lower priority program. This approach requires a complex executive but has inherent speed and flexibility.

6.0 Alternate Approaches

During the course of collecting data for this paper, other approaches to the problem of technical performance measurement for software developments have come to the attention of the author. They are presented here in summary fashion only in the interest of completeness and to demonstrate that many variations to the theme are possible.

6.1 Module Interface Control

Several categories of software modules can be defined and used in the construction of real-time systems. These include:

1. Application modules: Discrete software elements available for combination into application oriented packages (CPCIs);
2. Systems-operating modules: Discrete software elements which perform in the capacity of operating system components such as executives, priority schedulers, and generalized input/output systems;
3. User-interface modules: Customized user-interface software elements available for each application area to include the linkages necessary to combine the library modules and any specially developed software into a functioning system.

The concept of module interface control is that the complete interaction between modules be defined by the SPO or the contractor. This will include such areas as input/output data flows, computational speeds, etc.. These interface paramaters are then closely monitored

and controlled as the focus for technical performance measurement. The actual design and construction of the module is only considered when and if there is a change in the specified interfaces.

6.2 Library Utilization

One concept for improving the performance of programmers in software developments is to utilize a library which permanently records all software debugging activity either through compilation or through direct simulation of the target computer. The regular screening and reading of program errors will identify weak programmers to management for corrective action. This approach has its obvious limitations in terms of both the human factors aspect of the procedure and the difficulty associated with directing its implementation.

6.3 Independent Validation & Verification

For software developments with critical military applications, a parallel but independent software test and evaluation process can prove effective. There is precedent for such activity on programs such as Safeguard and Minuteman. There is evidence that improved software quality, and enhanced confidence in the functional capability of the software results. This approach has increased cost and schedule implications. Under certain circumstances however, it could be the only certain method of assuring software technical performance during the development process.

7.0 Summary and Conclusions

The design and development of the software portion of computer oriented system acquisitions is rapidly becoming a limiting factor in terms of cost, schedule and technical performance of the system. Although the software development life cycle is fairly well understood, there exists little or no definitive guidance regarding effective measurement of a contractor's progress toward achievement of technical performance goals. The historical application of hardware oriented program milestones has proved less than effective and both industry and Government are actively seeking improved management methods. One of the major problems in a large software development project is being able to portray an accurate picture of software status to upper management. This report has examined and proposed a two dimensional weighted software matrix which can be used to satisfy this need. This approach has been attempted on at least one large computer systems acquisition program and found to be effective in representing a realistic picture of software status. Another unique but complementory approach to monitoring the software development progress is to employ an integrated software engineering methodology. In this case, management attention is concentrated on fundamental micro and macro aspects of the development process which can be directly related to the system quantitative performance requirements. Recommended areas for SPO concentration include data structures, control structures, real-time processing, program and data organization, and memory allocation. These particular aspects were selected because they are absolutely fundamental in establishing the charactor and capability of the software sub-system.

A corollary benefit of selecting the above aspects for technical performance measurement, is that they will focus contractor management attention on the essentials of the software design process. It is concluded that there is significant room for improvement over current methods of measuring software technical performance during the acquisition cycle. The methods recommended in sections 4 and 5 of this paper can provide improved Program Office visibility into the software development cycle for both timely detection of problem areas and for enhanced confidence in a properly engineered and integrated system. The alternate approaches described in section 6 represent areas for further consideration with the concept of module interface control offering the most promise for future research.

ANNOTATED BIBLIOGRAPHY

1. John, Koudela, Jr., The Past, Present, and Future of Minicomputers: A Scenario, Proceedings of the IEEE, Vol.61, No.11, Nov 73

An article which traces the evolution of minicomputers from 1955 to 1973. Hardware, software, and trade offs are discussed.

2. Science Applications Inc., Special Presentation on Independent Software Test and Evaluation Technology, Bedford, Mass., Dec 73

Briefing material from the special presentation which describe the advantages of having an independent testing agency during the development of large and important software programs.

3. Air Force Systems Command, Project ACE-Findings and Action Plans, Andrews AFB, Md., Oct 73

The report of a high level Air Force Systems Command study group that investigated current acquisition methodology in the Air Force. Software definciencies are cited in two of the findings.

4. Herbert E. Pike, Jr., Software Production for Minicomputers, Proceedings of the IEEE, Vol.61, No.11, Nov 73

A comprehensive article which discusses the software development process to include analysis, design, implementation, integration, testing, delivery and maintenance. The viewpoint is basic and technically fundamental.

5. Martin L. Rubin, Handbook of Data Processing Management, Vol. I, New York, Brandon/Systems Press, 1970

A text which considers the software development cycle in eight distinct phases. Each phase is described in separate chapters from a management point of view.

6. B. W. Boehm, Software and It's Impact: A Quantitative Assesment, Datamation, May 73

An article which explores the role of software in computer systems. Strentghs and weaknesses of current programming methods are discussed.

7. Gerald M. Weinberg, The Psychology of Computer Programming, New York, Van Nostrand Reinhold, 1971

A text which explores the human behavior aspect of programming in a group or project environment. Provides unique and informative insight into the programmers world and the development process.

8. Akiro Sekino, Performance Evaluation of Multiprogrammed Time-Shared Computer Systems, MIT, Project MAC-TR-103, 1972

A doctoral dissertation which derives mathematical models to evaluate the performance of a large computer system currently in existance at MIT.

9. F. W. Vote, Multiprogramming Systems Evaluated Through Synthetic Programs, MIT Lincoln Laboratory, AD-771798, 1973

A chartered review of the use of simulation in the form of synthetic programs intended to emulate a real environment so as to assess the capabilities and limitations of large and complex computer systems.

DEPARTMENT OF THE AIR FORCE
Headquarters US Air Force
Washington DC 20330

DRAFT

AFR 800-14
21 January 1974

Acquisition Management

MANAGEMENT OF COMPUTER RESOURCES IN SYSTEMS

This regulation establishes policy for the acquisition and support of computer equipment and computer programs employed as dedicated elements, subsystems or components of systems developed or acquired under the program management concept established in AFR 800-2. It supplements AFR 800-3 and complements AFR 300-1. Computer resources in all other systems will be managed in accordance with AFRs 71-11, 80-2, 100-2, 102-5 and 300-2 as applicable.

OPR: RDM

DISTRIBUTION: S;X (Defense Systems Management School, Ft. Belvoir, VA 20060 ----------100)

DRAFT

1. Applicability of This Regulation. This regulation applies to all Air Force activities with responsibilities for planning, developing, acquiring, supporting, and using systems managed and acquired under AFR 800-2.

2. Objective of This Regulation. The objective of the Air Force is to insure that systems' computer resources are planned, developed, acquired, employed, and maintained to accomplish Air Force assigned missions effectively, efficiently, and economically.

3. Air Force Policy on Management of Computer Resources In Systems.

a. Computer resources in systems will be managed as elements or subsystems of major importance during conceptual, validation, full-scale development, production, deployment, operation and support phases. Requirements will be allocated to subsystems on a realistic and meaningful basis using in-depth trade-off studies and cost-effectiveness analyses.

b. Management responsibility for the integration of computer equipment and computer programs into a system will remain centralized for the life of the system. Development, maintenance and modification of selected computer programs may be decentralized commensurate with operational and support requirements.

c. In-house computer equipment maintenance and computer program development and maintenance capabilities will be established and used where economical or required to best satisfy system requirements. Common and existing capabilities will be used wherever practicable.

d. Computer equipment and computer programs will be standardized to the extent practicable within each system as well as across systems.

e. Automatic data processing (ADP) standards and higher level programming languages will be used or established in the system under development to the maximum extent practicable.

f. Computer equipment and computer program trade-offs will be conducted throughout the life cycle of the system to minimize cost and insure growth capability consistent with system needs.

g. Organizational responsibilities and computer resource requirements, including support facilities, documentation and other essential resources will be identified early in the system development program to insure coordinated actions and integrated support for the system life cycle.

h. Data item descriptions will be identified and developed (and approved as may be required by AFR 178-8) to insure timely and adequate program documentation support throughout the system life cycle.

i. Solicitation documents will include explicit statements establishing Air Force rights to computer programs required to operate and support the system. This includes those computer programs and associated documentation required for the maintenance and modification of these programs.

j. Configuration management procedures will be developed to assure firm configuration control during development, test, transition, operational maintenance, and major modification.

k. An inventory of computer equipment and computer programs will be developed and maintained.

l. Provision will be made for user involvement in computer program development, test, operational maintenance and major modification as system needs dictate.

m. Program Management Directives (PMDs) will require and Program Management Plans (PMPs) will provide for:

(1) Establishment of computer technical and managerial expertise, independent of the system prime or computer program development contractor, responsive to, preferably as an organic capability of, the Program Office (PO).

(2) The specification and allocation of system performance and interface requirements to be met by computer equipment and computer programs.

(3) Reliability, maintainability, and availability as prime development objectives.

(4) Computer equipment capacity and computer program design during the planning and development phases to provide for flexibility, growth and ease of modification and maintenance throughout the system life.

(5) The timely preparation of development, acquisition, operational maintenance and support plans for computer equipment, computer programs, supporting documentation and facilities covering the expected system life.

(6) Level of simulation to be employed to assist and assure the acquisition of systems responsive to mission requirements and to minimize the cost or risk associated with changes throughout the system life cycle.

(7) The comprehensive test, validation and verification of computer equipment and computer programs. Special emphasis will be directed to these items during the testing and evaluation conducted in accordance with AFR 80-14.

(8) The identification of computer equipment, computer programs and associated documentation as configuration items (CIs).

(9) Work breakdown structures (See MIL STD-881) designed to facilitate identification of computer resource costs.

(10) Coverage of computer equipment and computer programs during the conduct of system design reviews, audits, and management assessments.

4. Headquarters USAF Responsibilities.

a. Insures that the management of computer resources in systems is consistent with the policies contained in this and other applicable regulations.

b. Insures that policies and procedures for the management of computer equipment and computer programs are consistent with other applicable policies, regulations and directives.

5. AFSC Responsibilities.

a. Provides for the implementation of this regulation in the development, acquisition, turnover and transition of systems involving computer resources.

b. Maintains an organic capability of computer technical and managerial expertise.

c. Provides for the standardization of computer equipment and computer programs between and within systems and insures optimum usage of available computer resources as practicable.

d. Insures the timely application of advanced computer technology into systems.

e. Develops jointly with AFLC an inventory and data base on computer equipment and computer programs used in systems. Provides update

information to AFLC as new systems are developed.

f. Satisfies other responsibilities as defined in the Air Force 800 series and other applicable regulations.

6. Program Manager Responsibilities.

a. Provides management and technical emphasis to computer equipment and computer program requirements identified in the PMD.

b. Directs the preparation, update and implementation of the PMP consistent with the policies of this regulation.

c. Insures that the PO works with AFLC and the user to incorporate their needs into the PMP, supporting plans and other system documents prepared and implemented by the PO.

d. Satisfies other responsibilities as defined in the Air Force 800 series and other applicable regulations.

7. AFLC Responsibilities.

a. Provides for the implementation of this regulation during the transition and support of systems involving computer resources.

b. Maintains an organic capability of computer technical and managerial expertise.

c. Develops jointly with AFSC and maintains an inventory and data base on computer equipment and computer programs in systems.

d. Participates with the PO and the user in the preparation, update and implementation of the PMP, support plans (including the integrated logistics system plan) and other system documents.

e. Determines, in conjunction with the PO and the users, responsibilities for the maintenance and modification of computer equipment and computer programs and insures incorporation of these responsibilities in the appropriate system documents.

f. Programs for, establishes and operates facilities determined necessary to support AFLC assigned responsibilities for the integration and maintenance of the total system. Common and existing facilities will be used wherever practicable.

g. Insures the standardization of computer equipment and computer program support facilities and equipment wherever practicable.

h. Satisfies other responsibilities as defined in the Air Force 800 series and other applicable regulations.

8. Using Activities Responsibilities.

a. Provide for the implementation of this regulation in the turnover, operation and maintenance of systems involving computer resources.

b. Maintain an organic capability of computer technical and managerial expertise.

c. Participate with the PO and AFLC in the preparation, update and implementation of the PMP, support plans and other system documents. Insure accurate incorporation and timely update, within the approved program, of mission requirements.

d. Determine, in conjunction with the PO and AFLC, responsibilities for the maintenance and modification of computer equipment and modifications to computer programs and insure incorporation of these responsibilities in the appropriate system documents.

e. Program for, establishe and operate facilities determined necessary to support assigned responsibilities for the maintenance, modification and development of computer programs.

f. Satisfy other responsibilities as defined in the Air Force 800 series and other applicable regulations.

9. Air Training Command Responsibilities.

a. Reviews system documents and initiates training support planning.

b. Provides and administers training programs to support systems in accordance with AFR 50-9.

10. Air University Responsibilities.

Provides professional education in computer sciences and management.

Explanation of Terms

1. Availability - See MIL STD-721.
2. Computer Resources - the totality of computer equipment, computer programs, associated documentation, contractual services, personnel and supplies.
3. Computer Program - See AFR 300-2.
4. Configuration Item - See AFR 65-3.
5. Data Item Description - See AFR 300-10.
6. Higher Order Programming Language - See AFR 300-10.
7. Transition - See AFR 800-4.
8. Turnover - See AFR 800-4.
9. Program Manager - See AFR 800-2.
10. Program Management Directive - See AFR 800-2.
11. Program Office - See AFR 800-2.
12. Program Management Plan - See AFR 800-2.
13. Simulation - The representation of physical systems or phenomena by computers, models or other equipment.
14. Validation (of computer equipment and computer programs). The process of determining if the computer equipment and computer program(s) were developed in accordance with the stated specifications.
15. Verification - The process of determining if the computer equipment and computer program(s) satisfactorily performs, in the mission environment, the function(s) for which it was designed.

ATTACHMENT #1

ATTACHMENT # 2

CPCI	Wgt	Design	Code & Checkout	Verify
1. Special File Maintenance	5	50%	30%	20%
		80%	10%	
2. Space Weapons Support	3	30%	35%	35%
		60%	20%	
3. Catalog Data Applications	10	40%	30%	30%
		63%	25%	
4. Special Events CPCI	10	40%	25%	35%
		65%	minimal	
5. SCC EXEC/MMI	20	45%	25%	30%
		20%	15%	
6. System Support - Training & CAI	9	55%	25%	20%
		70%	55%	
7. SCC Unique Math Routines	10	20%	50%	30%
		75%	45%	
8. Orbit Prediction	12	30%	30%	40%
		65%	minimal	

wgt →

81% complete

Working Copy

CPCI	Wgt	Design	Code & Checkout	Verify
9. Orbit Optimization	10	40%	30%	30%
		65%	minimal	
10. Observation Collection	11	60%	15%	25%
		65%	minimal	

NCS	Wgt	Part II Design	Code & Checkout	Verify
Ops	70	40%	30%	30%
		65%	40%	0
Support	20	40%	35%	25%
		75%	50%	0
Utility	10	35%	45%	20%
		98%	75%	20%
		.38	.37	.25
		.78	.55	.07

Working Copy

Design Code/Checked Validated

NCS/SCC SUMMARY

25.8 + 5.1 + 0 = 30.9

SCC	wgt	.41	.30	.29
	comp	.63	.17	0
NCS	wgt	.38	.37	.25
	comp	.78	.55	.07

29.6 + 20.4 + 1.75 = 51.8

Wgt = Weight

Comp = Complete to date

Working Copy

Working Copy

ATTACHMENT #3

Glossary of Digital-Computer Terms

This glossary was provided by the Guest Editor for the convenience of readers of this Special Issue.

absolute: Pertaining to an address fully defined by a memory address number, or to a program which contains such addresses (as opposed to one containing symbolic addresses).

accumulator: A register in which numbers are totaled, manipulated, or temporarily stored for transfers to and from memory or external devices.

ADD: Restrictive: "two's complement" addition of binary numbers. General: any arithmetic addition.

address: (noun) A number which identifies one location in memory. (verb) To direct the computer to read a specified memory location (synonymous with "reference").

address modification: A programming technique of changing the address specified by a memory-reference instruction, so that each time that particular instruction is executed, it will affect a different memory location.

address word: A computer word which contains only the address of a memory location.

Algol: (Algebraic-Oriented Language) An international algebraic procedural language for a computer programming system.

algorithm: A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.

alphanumeric: Pertaining to a character set that contains both letters and numerals, and usually other characters.

alter: A modification of the contents of an accumulator or extend bit, e.g., clear, complement, or increment.

AND: A logical operation in which the resultant quantity (or signal) is true if all of the input values are true, and is false if at least one of the input values is false.

argument: 1) A variable or constant which is given in the call of a subroutine as information to it. 2) A variable upon whose value the value of a function depends. 3) The known reference factor necessary to find an item in a table or array, i.e., the index.

arithmetic logic: The circuitry involved in manipulating the information contained in a computer's accumulators.

arithmetic operation: Restrictive: a mathematical operation involving fundamental arithmetic (addition, subtraction, multiplication, division), specifically excluding logical and shifting operations. General: any manipulation of numbers.

array: A set of lists of elements, usually variables or data.

ASCII: An abbreviation for American Standard Code for Information Interchange.

assemble: To translate from a symbolic program to a binary program by substituting binary operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

assembler: A program for a computer which converts a

program prepared in symbolic form (i.e., using defined symbols and mnemonics to represent instructions, addresses, etc.) to binary machine language.

assembly language: The source language used as input to an assembler and translated by the assembler into machine language.

auxiliary storage: Storage that supplements core memory, such as disk tape.

background processing: The automatic execution of a low-priority computer program when higher priority programs are not using the system resources.

base: The quantity of different digits used in a particular numbering system. The base in the binary numbering system is two; thus there are two digits (0 and 1). In the decimal system (base 10), there are ten digits (0 through 9).

base address: A given address from which an absolute address is derived by combination with a relative address.

base page: The lowest numbered page of a computer's memory. It can be directly addressed from any other page.

binary: Denoting the numbering system based on the radix two. Binary digits are restricted to the values 0 and 1.

binary-coded decimal (bcd): coding method for representing each decimal digit (0–9) by specific combinations of four bits. For example, the 8–4–2–1 bcd code commonly used with computers represents "1" as 0001 and "9" as 1001.

binary program: A program (or its recording form) in which all information is in binary machine language.

bistable: Pertaining to an electronic circuit having two stable states, controllable by external switching signals, analogous to an on–off switch.

bit (b): A single digit in a binary number, or in the recorded representation of such a number (by hole punches, magnetic states, etc). The digit can have one of only two values, 0 or 1.

bit density: A physical specification referring to the number of bits which can be recorded per unit of length or area.

bit serial: One bit at a time, as opposed to bit parallel, in which all bits of a character can be handled simultaneously.

block: A set of consecutive machine words, characters, or digits handled as a unit, particularly with reference to I/O.

bootstrap: A technique or device designed to bring itself into a desired state by means of its own action, e.g., a routine whose first few instructions are sufficient to bring the rest of itself into the computer from an input device.

branch: A point in a routine where one of two or more choices is made under control of the routine.

breakpoint: A point in a computer program at which conditional interruption is made to permit visual check, printouts, or other debugging aids.

buffer: A register used for intermediate storage of information in the transfer sequence between the computer's accumulators and a peripheral device or a designated area of memory used to temporarily hold data.

bug: A mistake in the design or implementation of a program resulting in erroneous results.

bulk memory: Storage in addition to the main memory of the computer, e.g., magnetic tape, disk, or drum.

bus: A major electrical path connecting two or more electrical circuits.

byte: A group of binary digits usually operated upon as a unit, frequently 8 b.

calling sequence: A specified set of instructions and data necessary to set up and call a given routine.

carry: A digit, or equivalent signal, resulting from an arithmetic operation which causes a positional digit to equal or exceed the base of the effective numbering system.

central processing unit: The unit of a computing system that includes the circuits controlling the interpretation and execution of instructions—the computer proper, excluding I/O and other peripheral devices.

character: The general term to include all symbols such as alphabetic letters, numerals, punctuation marks, mathematical operators, etc. Also, the coded representation of such symbols.

checkpoint: A point in time during a program run at which processing is momentarily halted to make a record, on an external storage medium, of the condition of the variables of the program being executed.

clear: To erase the contents of a storage location by replacing the contents, normally with zeros or spaces; to set to zero.

code: A system of symbols which can be used by machines, such as a computer, and which in specific arrangements have a special external meaning.

coding: Writing instructions for a computer using symbols meaningful to the computer, or to an assembler, compiler, or other language processor.

compatibility: The ability of an instruction or source language to be used on more than one computer.

compile: To produce a binary-coded program from a program written in source (symbolic) language, by selecting appropriate subroutines from a subroutine library, as directed by the instructions or other symbols of the source program. The linkage is supplied for combining the subroutines into a workable program, and the subroutines and linkage are translated into binary code.

compiler: A language translation program, used to transform symbols meaningful to a human operator into codes meaningful to a computer. More restrictively, a program which translates a machine-independent source language into the machine language of a specific computer, thus excluding assemblers.

complement: (one's) To replace all 0 bits with 1 bits and vice versa. (two's) To form the one's complement and add 1.

computation: The processing of information within the computer.

computer (digital): An electronic instrument capable of accepting, storing, and arithmetically manipulating information, which includes both data and the controlling program. The information is handled in the form of coded binary digits (0 and 1), represented by dual voltage levels, magnetic states, punched holes, etc.

computer word: See "word."

conditioned assembly: Assembly of certain parts of a symbolic program only if certain conditions have been met.

configuration: The arrangement of either hardware instruments or software routines when combined to operate as a system.

console: Usually the external front side of a device, where controls and indicators are available for manual operation of the device.

constant: Numeric data used but not changed by the program.

contents: The information stored in a register or a memory location.

convert: 1) To change numeric data from one radix to another. 2) To transfer data from one recorded format to another.

core: The smallest element of a core-storage memory module.

It is a ring of ferrite material that can be magnetized in clockwise or counterclockwise directions to represent the two binary digits, 0 and 1.

core memory: The main high-speed storage of a computer, in which binary data are represented by the switching polarity of magnetic cores.

current location counter: A counter kept by an assembler to determine the address assigned to an instruction or constant being assembled.

current page: The memory page comprising all those locations which are on the same page as a given instruction.

cycle time: The length of time it takes the computer to reference one word of memory.

data: A general term used to denote any or all facts, numbers, letters, and symbols. It connotes basic elements of information which can be processed or produced by a computer.

data acquisition: The gathering, measuring, digitizing, and recording of continuous-form (analog) information.

data reduction: The transformation of raw information gathered by measuring or recording equipment into a more condensed, organized, or useful form.

data word: A computer word consisting of a number, a fact, or other information which is to be processed by the computer.

debug: To check for and correct errors in a program.

decimal: Denoting the numbering system based on the radix ten.

decrement: To change the value of a number in the negative direction. If not otherwise stated, a decrement by one is usually assumed.

delimiter: A character that separates, terminates, and organizes elements of a statement or programs.

device: An electronic or electromechanical instrument. Most commonly implies measuring, reading, or recording equipment.

diagnostic: (adjective) Relating to test programs for detection of errors in the functining of hardware or software, or the messages resulting from such tests. (noun) The test program or message itself.

digit: A character used to represent one of the nonnegative integers smaller than the radix, e.g., in binary notation, either 0 or 1.

direct address: An address that specifies the location of an instruction operand.

direct memory access: A means of transferring a block of information words directly between an external device and the computer's memory, bypassing the need for repeating a service routine for each word. This method greatly speeds the transfer process.

disable: A signal condition which prohibits some specific event from proceeding.

disk storage: A means of storing binary digits in the form of magnetized spots on a circular metal plate coated with a magnetic material. The information is stored and retrieved by READ–WRITE heads which may be positioned over the surface of the disk either by moving the heads or the disk itself.

documentation: Manuals and other printed materials (tables, listings, diagrams, etc.) which provide instructive information for usage and maintenance of a manufactured product, including both hardware and software.

double-length word: A word which, due to its length, requires two computer words to represent it. Double-length words are normally stored in two adjacent memory locations.

double-precision: Pertaining to the use of two computer words to represent one number.

downtime: The time interval during which the device is inoperative.

dummy: Used as an adjective to indicate an artificial address, instruction, or record of information inserted solely to fulfill prescribed conditions, as in a "dummy" variable.

dump: To copy the contents of all or part of core memory, usually onto an external storage medium.

dynamic relocation: The ability to move programs or data from auxiliary memory into main memory at any convenient location. Normally the addresses of programs and data are fixed when the program is compiled.

effective address: The address of a memory location ultimately affected by a memory reference instruction. It is possible for one instruction to go through several indirect addresses to reach the effective address.

enable: A signal condition which permits some specific event to proceed, whenever it is ready to do so.

EXCLUSIVE-OR: A logical operation in which the resultant quantity (or signal) is true if at least one (but not all) of the input values is true, and is false if the input values are all true or all false.

execute: To fully perform a specific operation, such as would be accomplished by an instruction or a program.

exit sequence: A series of instructions to conclude operation in one area of a program and to move to another area.

external storage: A separate facility or device on which data usable by the computer are stored (such as paper tape, tape, or disk).

field: 1) One or more characters treated as a unit. 2) A specified area of a record used for a single type of data.

file: A collection of related records treated as a unit.

file name: Alphanumeric characters used to identify a particular file.

fixed point: A numerical notation in which the fractional point (whether decimal, octal, or binary) appears at a constant predetermined position. (Compare with "floating point.")

flag: A variable or register used to record the status of a program or device—in the latter case, sometimes called "device flag."

flip-flop: An electronic circuit having two stable states, and thus capable of storing a binary digit. Its states are controlled by signal levels at the circuit input and are sensed by signal levels at the circuit output.

floating point: A numerical notation in which the integer and the exponent of a number are separately represented (frequently by two computer words), so that the implied position of the fractional point (decimal, octal, or binary) can be freely varied with respect to the integer digits. (Compare with "fixed point.")

flow chart: A diagram representing the operation of a computer program.

foreground processing: Higher priority processing that takes precedence over "background processing" and can interrupt such processing. It results from real-time events or enquiries.

format: A predetermined arrangement of bits and characters.

Fortran: A programming language (or the compiler which

translates this language) which permits programs to be written in a form resembling algebra, rather than in detailed instruction by instruction format.

forward referencing: The need to refer to a symbol in a program prior to its definition (i.e., trying to assemble the instruction JUMP PLACE, where PLACE is a location symbol further down in the program code).

full-duplex: Describing a communication channel capable of simultaneous and independent transmission and reception.

gate: An electronic circuit capable of performing logical functions such as AND, OR, NOR, etc.

half-duplex: Describing a communication channel capable of transmission and/or reception, but not both simultaneously.

hardware: Electronic or electromechanical components, instruments, or systems.

high core: Core-memory locations having high-numbered addresses.

INCLUSIVE-OR: A logical operation in which the resultant quantity (or signal) is true if at least one of the input values is true, and is false if the input values are all false.

increment: To change the value of a number in the positive direction. If not otherwise stated, an increment by one is usually assumed.

incremental magnetic tape: A form of magnetic tape recording in which the recording transport advances by small increments (e.g., 0.005 in), stopping the tape advancement long enough to record one character at the spot located under the recording head.

index register: A memory device containing an index. (See "address modification.")

indirect address: The address initially specified by an instruction when it is desired to use that location to redirect the computer to some other location to find the "effective address" for the instruction.

information: A unit or set of knowledge represented in the form of discrete "words," consisting of an arrangement of symbols or (so far as the digital computer is concerned) binary digits.

inhibit: To prevent a specific event from occurring.

initialize: The procedure for setting various parts of a stored program to starting values, so that the program will behave the same way each time it is repeated. The procedures are included as part of the program itself.

input: Information transferred from a peripheral device into the computer. Also applied to the transfer process itself.

input/output (I/O): Relating to the equipment or method used for transmitting information into and out of the computer.

input/output channel: The complete input or output facility for one individual device or function, including its assigned position in the computer, the interface circuitry, and the external device.

instruction: A written statement or the equivalent computer-acceptance code, which tells the computer to execute a specified single operation.

instruction code: The arrangement of binary digits which tells the computer to execute a particular instruction.

instruction logic: The circuitry involved in moving binary information between registers, memory, and buffers in prescribed manners, according to instruction codes.

instruction word: A computer word containing an instruction code. The code bits may occupy all or (as in the case of memory reference instruction words) only part of the word.

interface: The connecting circuitry which links the central processor of a computer system to its peripheral devices.

internal storage: The storage facilities forming an integral physical part of the computer and directly controlled by the computer. Also called "main memory" and "core memory."

interpreter: A program which translates and executes source language statements at run time.

interrupt: The process, initiated by an exteranl device, which causes the computer to interrupt a program in progress, generally for the purpose of transferring information between that device and the computer.

interrupt location: A memory location whose contents (always an instruction) are executed upon interrupt by a specific device.

iteration: Repetition of a group of instructions.

job: A unit of code which solves a problem, i.e., a program and all its related subroutines and data.

jump: An instruction which breaks the strict sequential location-by-location operation of a program and directs the computer to continue at another specified location anywhere in memory.

K: One thousand and twenty-four. For example, 4 K words of memory means 4096 words.

label: Any arrangement of symbols, usually alphanumeric, used in place of an absolute memory address in computer programming.

language: The set of symbols, rules, and conventions used to convey information, either at the human level or at the computer level.

leader: The blank section of tape at the beginning of the tape.

least significant digit: The rightmost digit of a number.

library routine: A routine designed to accomplish some commonly used mathematical function and kept permanently available on a library program tape (e.g., Fortran Library).

line feed: The Teletype operation which advances the paper by one line.

line number: In source languages such as Focal, Basic, and Fortran, a number which begins a line of the source program for purposes of identification. A numeric label.

linkage: In programming, code that connects two separately coded routines.

list: 1) A set of items. 2) To print out a listing on the line printer or Teletype. 3) See "pushdown list."

literal: A symbol which defines itself.

load: To put information into (memory, a register, etc.). Also (e.g., loading tape), to put information medium into the appropriate device.

load time: That time at which an assembled program is placed in the computer and readied for execution.

loader: A program designed to assist in transferring information from an external device into a computer's memory.

location: A group of storage elements in the computer's memory which can store one computer word. Each such location is identified by a number ("address") to facilitate storage and retrieval of information in selectable locations.

logical operation: A mathematical process based on the principles of truth tables, e.g., AND, INCLUSIVE-OR, and EXCLUSIVE-OR operations.

logic diagram: A diagram which represents the detailed internal functioning of electronic hardware, using binary logic symbols rather than electronic component symbols.

logic equation: A written mathematical statement, using

symbols and rules derived from Boolean algebra. Specifically (hardware design), a means of stating the conditions required to obtain a given signal.

loop: A sequence of instructions in which the last instruction is a jump back to the first instruction.

low core: Core-memory locations having low-numbered addresses.

machine: Pertaining to the computer hardware (e.g., machine timing, machine language).

machine language: The form of coded information (consisting of binary digits) which can be directly accepted and used by the computer. Other languages require translation to this form, generally with the aid of translation programs (assemblers and compilers).

machine timing: The regular cycle of events in the operation of internal computer circuitry. The actual events will differ for various processes, but the timing is constant through each recurring cycle.

macro: An assembly-time facility that allows lines of text to be named and saved by the assembler, to be retrieved and modified by the substitution of text for dummy names in the saved text. The resulting modified text is assembled at the point of retrieval.

macroinstruction: An instruction, similar in binary coding to the computer's basic machine-language instructions, which is capable of producing a variable number of machine-language instructions.

magnitude: That portion of a computer word which indicates the absolute value of a number, thus excluding the sign bit.

mask: A bit pattern which selects those bits from a word of data which are to be used in some subsequent operation.

mass-storage: Pertaining to a device, such as tape or disk, which stores large amounts of data readily accessible to the central processing unit.

media conversion: The transferral of recorded information from one recording medium (e.g., punched paper tape, magnetic tape, etc.) to another recording medium.

memory: An organized collection of storage elements (e.g., ferrite cores) into which a unit of information consisting of a binary digit can be stored and from which it can later be retrieved. Also, a device not necessarily having individual storage elements, but which has the same storage and retrieval capabilities (e.g., magnetic disks).

memory cycle: That portion of the computer's internal timing during which the contents of one location of memory are read out (into the transfer register) and written back into that location.

memory module: A complete segment of core storage, capable of storing a definable number of computer words (e.g., 4096 or 8192 words). Computer storage capacity is incremental by modules and is frequently rounded off and abbreviated as "4K" (i.e., 4096 or approximately 4000 words), "8K" (8192 or 8000), "16K," etc.

memory protect: A means of preventing inadvertent alteration of a selectable segment of memory.

memory reference: The address of the memory location specified by a memory-reference instruction, i.e., the location affected by the instruction.

microinstruction: An instruction which forms part of a larger composite instruction.

minicomputer: A general term used to describe small computers. In this sense, small usually implies both the computer's physical size and its word size (data-path width). Most minicomputers are designed with a 16-b word size, but sizes from 8 to 18 b are considered in the minicomputer range.

monitor: An operating programming system which provides a uniform method for handling the real-time aspects of program timing, such as scheduling and basic input/output functions.

most significant digit: The leftmost nonzero digit.

multilevel indirect: Indirect addressing using two or more indirect addresses in sequence to find the effective address for the current instruction.

multiple-precision: Referring to arithmetic in which the computer, for greatest accuracy, uses two or more words to represent one number.

multiprocessing: Utilization of several computers or processors to logically or functionally divide jobs or processes and to execute them simultaneously.

multiprogramming: A system of execution of two or more programs kept in core at the same time. Execution cycles between the programs.

normalize: To adjust the exponent and fraction of a floating-point quantity so that the fraction appears in a prescribed format.

object programming: The binary coded program which is the output after translation from the source language; the binary program which runs on the computer.

octal: Denoting a numbering system based on the radix eight. Octal digits are restricted to the values 0 through 7.

octal code: A notation for writing machine-language programs with the use of octal numbers instead of binary numbers.

off-line: Pertaining to the operation of peripheral equipment not under control of the computer.

one's complement: A number so modified that the addition to the modified number and its original value, plus one, will equal an even power of two. A one's complement number is obtained mathematically by subtracting the original value from a string of 1's, and electronically by inverting the states of all bits in the number.

on-line: Pertaining to the operation of peripheral equipment under computer control.

operand: That which is effected, manipulated, or operated upon. The address or symbolic-name portion of an assembler instruction.

operating system: An integrated collection of routines for supervising the sequencing of programs by a computer, e.g.: debugging, input/output, operation, compilation, and storage assignment.

operation (OP) code: That part of an instruction designating the operation to be performed.

operator: That symbol or code which indicates an action (or operation to be performed.

optimum code: A set of machine language instructions which is *particularly efficient with regard to a particular aspects*, e.g., minimum time to execute or minimum or efficient use of storage space.

OR: (inclusive) A logical operation such that the result is true if either or both operands are true, and false if both operands are false. (exclusive) A logical operation such that the result is true if either operand is true, and false if both operands are either true or false.

origin: The absolute address of the beginning of a section of code.

output: Information transferred from the computer to a peripheral device. Also applied to the transfer process itself.

overflow: A condition that occurs when a mathematical operation yields a result whose magnitude is larger than the program is capable of handling.

overlay: The operation of bringing into main memory and executing a segment which is a subprogram (i.e., a more or less separate entity) of a larger program.

packed word: A computer word containing two or more independent units of information. This is done to conserve storage when information requires relatively few bits of the computer word.

page: An artificial division of memory consisting of a fixed number of locations, dictated by the direct addressing range of memory reference instructions.

page zero: The memory page which includes the lowest numbered memory addresses.

parity bit: A supplementary bit added to an information word to make the total number of one bits always odd or even. This permits checking the accuracy of information transfers.

pass: The complete process of reading a set of recorded information (one tape, one set of cards, etc.) through an input device, from beginning to end.

patch: To modify a routine in a rough or expedient way.

peripheral device: An instrument or machine electrically connected to the computer, but which is not part of the computer itself.

plane: An arrangement of ferrite cores on a matrix of control and sensing wires. Several planes stacked together form a "memory module."

pointer address: Address of a core-memory location containing the actual (effective) address of desired data.

power-failure control: A means of sensing primary power failure so that a special routine may be executed in the finite period of time available before the regulated dc supplies discharge to unusable levels. The special routine may be used to preserve the state of a program in progress, or to shut down external processes.

priority: The automatic regulation of events so that chosen actions will take precedence over others in cases of timing conflict.

procedure: The course of action taken for the solution of a problem; also called "algorithm."

process control: Automatic control of manufacturing processes by use of a computer.

processor: The central unit of a computer system (i.e., the device which accomplishes the airthmetic manipulations), exclusive of peripheral devices. Frequently (when used as adjective) also excludes interface components, even though normally contained within the processor unit; thus "processor" options exlcude interface ("input/output") options.

program: A plan for the solution of a problem by a computer, consisting of a sequence of computer instructions.

program listing: A printed record (or equivalent binary-output program) of the instructions in a program.

programmer: A person who writes computer programs. Also (hardware), an interface card or instrument which sets up (or "programs") the various functions of one measuring instrument.

programming: The process of creating a program.

pseudoinstruction: A symbolic statement, similar to assembly-language instructions in general form, but meaningful only to the program containing it, rather than to the computer as a machine instruction.

punched tape: A string of tape, usually paper, on which information is represented by coded patterns of holes punched in columns across the width of the tape. There are commonly 8 hole positions (channels) across the tape.

pushdown list: A list that is constructed and maintained so that the next item to be retrieved is the item most recently sorted in the list.

queue: A waiting list. In time sharing, the monitor maintains a queue of user programs waiting for processing time.

radix: The base of a number system, the number of digit symbols required by a number system. See "binary," "octal."

random-access: Pertaining to a storage device in which the accessibility of data is effectively independent of the location of the data (synonomous with "direct-access").

read: The process of transferring information from an input device into the computer. Also, the process of taking information out of the computer's memory. (See "memory cycle.")

real time: The time elapsed between events occurring externally to the computer. A computer which accepts and processes information from one such event and is ready for new information before the next event occurs is said to operate in a "real-time environment."

record: A collection of related items of data, treated as a unit.

recursive subroutine: A subroutine capable of calling itself and returning at some later point to the program which initially called it.

reentrant code: A program segment (e.g., subroutine) which can be executed (i.e., reentered) by more than one other program simultaneously. This mode of operation requires a separate storage area for storing information that varies for each instance of execution.

register: An array of hardware binary circuits (flip-flops, switches, etc.) for temporary storage of information. Unlike mass-storage devices such as memory cores, registers can be wired to permit flexible control of the contained information, for arithmetic operations, shifts, transfers, etc.

relative address: The number that specifies the difference between the actual address and a base address.

relocatable: Pertaining to programs whose instructions can be loaded into any stated area of memory.

relocating loader: A computer program capable of loading and combining relocatable programs (i.e., programs having symbolic rather than absolute addresses).

reset: A signal condition representing a binary "zero."

response time: The time between initiating some operation from a terminal and obtaining results. Includes transmission time to the computer, processing time, access time to file records needed, and transmission time back to the terminal.

restart: To measure the execution of a program.

rotate: A positional shift of all bits in an accumulator (and possibly an extend bit as well) with those bits lost off one end of the accumulator "rotated" around to enter vacated positions at the other end.

routine: A program or program segment designed to accomplish a single function.

run time: The time in which a progran is executed.

segment: 1) That part of a long program which may be

resident in core at any one time. 2) To divide a program as in 1), or into two or more segments, or to store part of a program or routine on an external storage device to be brought into core as needed.

serial-access: Pertaining to the sequential or consecutive transmission of data to or from core, for example, paper tape. Contrast with "random-access."

service routine: A sequence of instructions designed to accomplish the transfer of information between a particular device and the computer.

set: A signal condition representing a binary "one."

shift: Restrictive (arithmetic shift): to multiply or divide the magnitude portion of a word by a power of two, using a positional shift of these bits. General: any positional shift of bits.

sign: The algebraic plus or minus indicator for a mathematical quantity. Also, the binary digit or electrical polarity representing such an indicator.

significant digit: A digit so positioned in a numeral as to contribute a definable degree of precision to the numeral. In conventional written form, the most significant digit in a numeral is the leftmost digit, and the least significant digit is the rightmost digit.

simulate: To represent the functioning of a device, system, or computer program with another system or program.

skip: An instruction which causes the computer to omit the instruction in the immediately following location. A skip is usually arranged to occur only if certain specified conditions are sensed and found to be true, thus allowing various decisions to be made.

snapshot dump: A dynamic printout during execution, at breakpoints and checkpoints, of selected areas in storage.

software: Computer programs. Also, the tapes or cards on which the programs are recorded.

software package: A complete collection of related programs, not necessarily combined as a single entity.

source program: A program (or its recorded form) written in some programming language other than machine language and thus requiring translation. The translated form is the "object program."

starting address: The address of a memory location in which is stored the first instruction of a given program.

statement: An instruction in any computer-related language other than machine language.

storage allocation: The assignment of blocks of data and instructions to specified blocks of storage.

storage capacity: The amount of data that can be entered, retained, and retrieved.

storage device: A device in which data can be entered, retained, and retrieved.

store: To put information into a memory location, register, or device capable of retaining the information for later access.

string: A connected sequence of entities, such as characters in a command string.

subroutine: A sequence of instructions designed to perform a single task, with provisions included to allow some other program to cause execution of the task sequence as if it were part of its own program.

subscript: A value used to specify a particular item in an array.

swapping: In a time-sharing environment, the action of either temporarily bringing a user program into core or storing it on the disk or other system device.

switch: A device of programming technique for making selections.

symbol table: A table in which symbols and their corresponding values are recorded.

symbolic address: A label assigned in place of absolute numeric addresses, usually for purposes of relocation. (See "relocatable.")

symbolic coding: Broadly, any coding or programming system in which symbols other than actual machine operations and addresses are used.

symbolic instruction: An instruction which is the basic component of an assembly language (input to assembler) and is directly translatable into machine language.

syntax: 1) The structure of expressions in a programming language. 2) The rules governing the structure of a programming language.

table: A collection of data stored for ease of reference, generally an array.

temporary storage: Storage locations reserved for immediate results.

terminal: A peripheral device in a system through which data can either enter or leave the computer.

time sharing: A method of allocating central-processor time and other computer services to multiple users so that the computer, in effect, processes a number of programs simultaneously.

time slicing: A method of job scheduling in a multiprogrammed system. This refers to the allocation of fixed amounts of computing time among users on a round-robin basis. Interrupts are generated by a fixed interval timer causing control to pass to the next waiting service request.

toggle: (adjective) Using switches to enter data into the computer memory.

transfer vector: A table, usually at a fixed location in memory, containing jump instructions and/or indirect addresses for jump instructions. A jump to a particular routine or the address of the routine is placed in a particular place in the table. Other routines can call this routine without necessarily knowing the actual location of this routine in memory. This technique is used frequently when a relocatable assembler is not available for a particular machine.

truncation: The reduction of precision by dropping one or more of the least significant digits; e.g., 3.141592 truncated to 4 decimal digits is 3.141.

truth table: A table listing of all possible configurations and resultant values for any given Boolean algebra function.

two's complement: A number so modified that the addition of the modified number and its original value will equal an even power of two. Also, a kind of arithmetic which represents negative numbers in two's-complement form so that all addition can be accomplished in only one direction (positive incrementation). A two's-complement number is obtained mathematically by subtracting the original value from an appropriate power of the base two, and electronically by inverting the states of all bits in the number and adding one (complement and increment).

underflow: A condition that occurs when a floating-point operation yields a result whose magnitude is smaller than the program is capable of handling.

updated program: A program to which additions, deletions, or corrections have been made.

user: The person or persons who program and operate a particular computer.

utility routine: A standard routine to assist in the operation of the computer (e.g., device drivers, sorting routines, etc.) as opposed to mathematical ("library") routines.

variable: A symbol whose value changes during the execution of a program.

waiting loop: A sequence of instructions (frequently only two) which are repeated indefintely until a desired external event occurs, such as the receipt of a flag signal.

word: A set of binary digits handled by the computer as a unit of information. Its length is determined by hardware design, e.g., the number of cores per location and the number of flip-flops per register.

word length: The number of bits in a word.

working register: A register whose contents can be modified under control of a program. Thus a register consisting of manually operated switches is not considered a working register.

write: The process of transferring information from the computer to an output device. Also, the process of storing (or restoring) information into the computer's memory. (See "memory cycle.")